

Réalisation d'un système solaire

version 1

Interface Homme-Machine : Unity (Version enseignant)

Voici les objectifs de ce sujet :

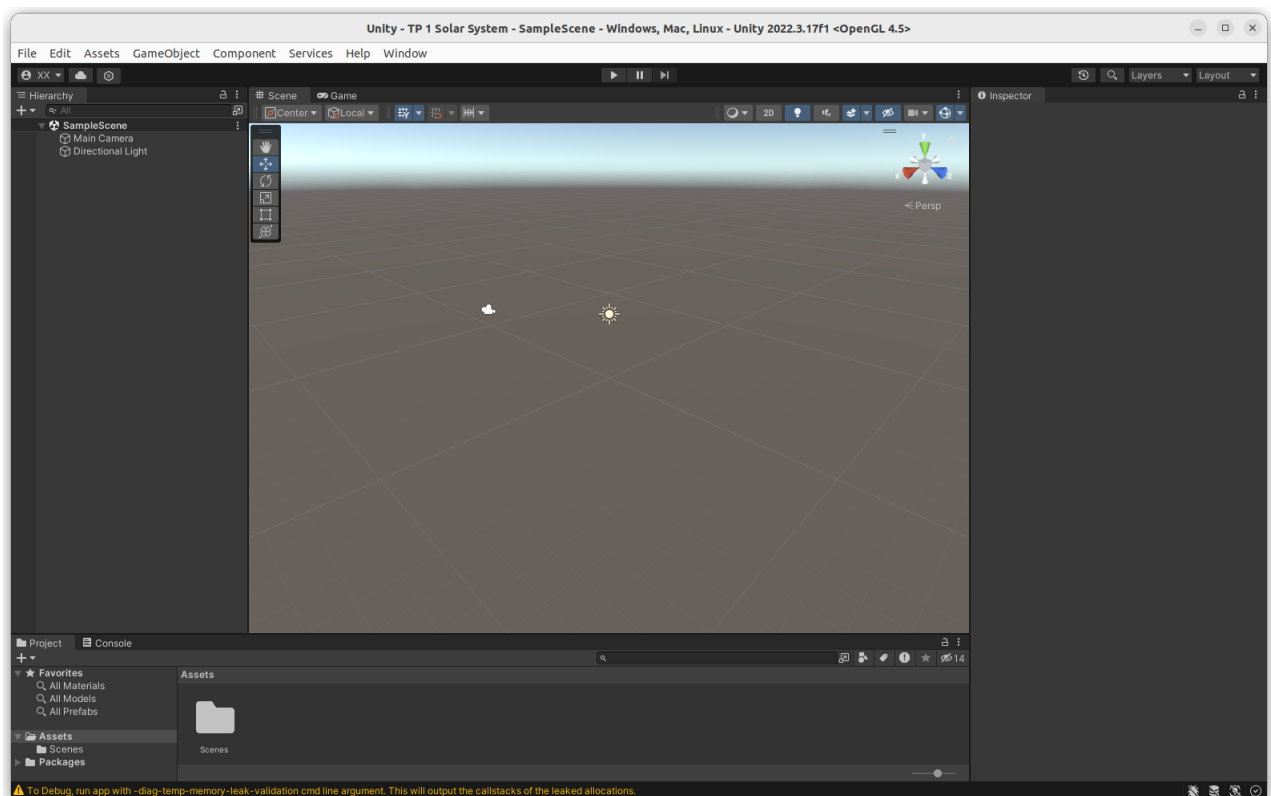
- Comprendre l'IDE **Unity**.
- Création d'un projet.
- Placer des objets dans l'environnement.
- Comprendre la hiérarchie pour différencier transformation locale et globale.
- Comprendre le canevas 2D.
- Manipuler des widgets classiques.
- Exploiter les événements pour ajouter des interactions.
- Réaliser des scripts simples pour l'animation.

Solution

1 Création du projet

1. Installer **UnityHub** et la dernière version de **Unity**. La version sur laquelle ce tutorial a été réalisée est : **UnityHub 3.7.0** et **Unity 2023.3.17f1**.
2. Ouvrir **UnityHub** :
 - "New Project"
 - "3D Core"
 - "Project name" → "TP1 Solar System"
 - "Location" → choisir le dossier de destination
 - "Create Project"

Une nouvelle scène est générée et l'interface affichée en figure 1a s'affiche.



(a) Nouvelle scène **Unity**

Solution

La fenêtre principale contient deux onglets :

- **Scene** permet de vérifier les propriétés des objets.
- **Game** affiche la vue de la caméra au démarrage de l'application.

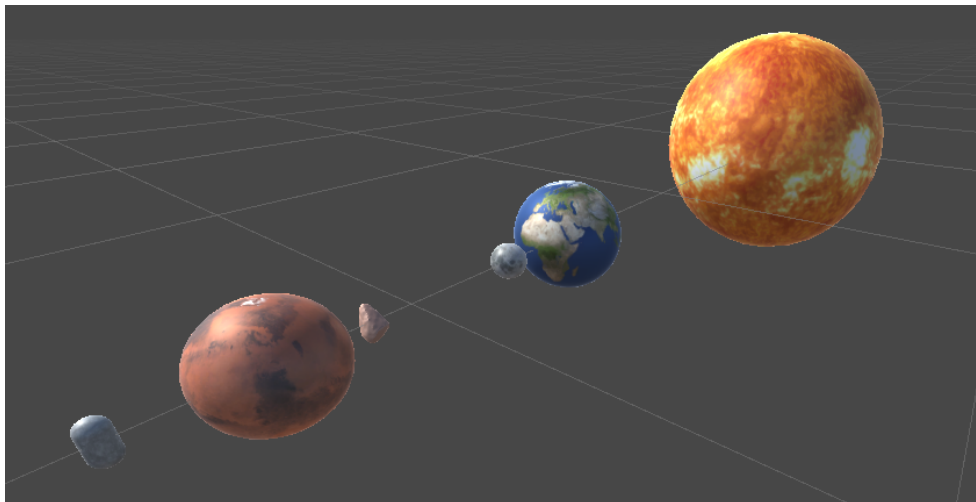
Attention à bien vérifier que **Scene** est sélectionné pour mettre en place la scène, sinon on a l'impression que l'interaction est bloquée. Un moyen de s'en assurer est de sauvegarder la scène : si le mode **Play** ou **Pause** (voir les icônes en haut de l'interface) a été activé, un message d'erreur apparaît.

Pour arrêter le mode **Play** ou **Pause**, il faut cliquer sur les icônes correspondantes pour qu'elles repassent en gris.

2 Description générale de l'application

Dans ce TP, nous allons modéliser un système solaire qui n'est pas réaliste physiquement. Pour cela, vous devez réaliser en vous inspirant ce qui a été fait en cours en utilisant la hiérarchie, un système solaire composé des éléments suivants :

- *Soleil* qui est au centre de notre animation.
- Deux planètes (disons *Terre* et *Mars*) qui tournent autour du *Soleil*.
- Des lunes pour chaque planète. Pour information, la *Terre* possède une *Lune* et *Mars* possède 2 lunes (nommées *Phobos* et *Deimos*).



L'objectif ici est de réaliser des manipulations d'Unity, vous pouvez/devez expérimenter les points suivants :

1. Utilisez la forme primitive Capsule pour Deimos.
2. Utilisez un maillage produit avec vos mains sous Blender ou *via* un objet quelconque (format `.obj` de préférence) trouvé sur l'Internet pour remplacer Phobos.
3. Mettez une texture sur les objets (pas forcément celle des planètes, mais ce que vous voulez).
4. Modifiez la "**skybox**" de votre caméra (plusieurs possibilités sont permises, mais utilisez bien la documentation et vos intuitions pour le faire, sans chercher en premier lieu une solution sur le net).
5. Placez les objets *de manière hiérarchique* pour répercuter les transformations de leur parent.

Vous pouvez utiliser les ressources mises à disposition sur **UPdago**. Depuis l'interface de **Unity** : repérer l'onglet **Project** et le dossier **Assets**, dans lequel vous glissez-déposez le dossier **Images**.

Solution

Le contenu du dossier **Images** et du sous-dossier **Materials** devrait apparaître sous forme d'icônes.

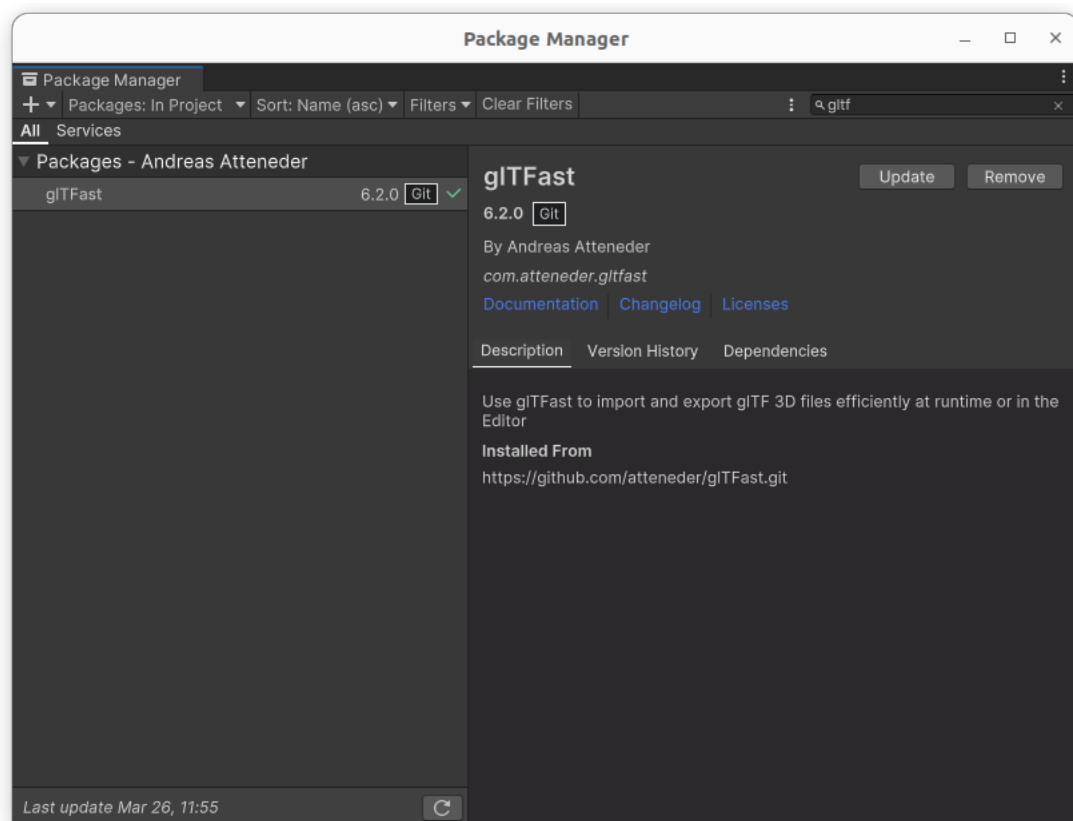
Il est possible que l'image **rosetta**, associée au fichier **Phobos_1_1000.glb** ne soit pas reconnue correctement : cela est dû à un format non supporté en natif par **Unity**. Deux solutions :

- chercher sur un site libre un objets 3D au format **OBJ**, par exemple sur le site TurboSquid ^a (mot-clé : "*rock*" avec filtre "*price = free*"). Une fois l'archive téléchargée, il faut l'ouvrir puis glisser-déposer le dossier correspondant dans **Assets** > **Images** ;

- installer un plugin spécifique pour reconnaître le format GLTF :
 - Dans l'interface, ouvrir **Window > Package Manager**.
 - Cliquer sur le bouton "+" en haut à gauche de la nouvelle fenêtre et sélectionner "Add package from git URL...".
 - Entrer un lien d'un outil d'import, par exemple `https://github.com/atteneder/gltFast.git` (Fig. 2a).
 - Unity importe le *package* et relit automatiquement les **assets** → `Phobos_1_1000.glb` est maintenant reconnue et on peut l'insérer directement dans la scène.

Noter que tous les assets sont automatiquement compilés et le résultat porte l'extension `.meta`.

a. Nécessite une inscription gratuite



(a) Import du package GLTFast

Solution

2.1 Construction du système solaire

L'onglet **Hierarchy** contient les objets définis par défaut avec la nouvelle scène : une caméra **Main Camera** et une source de lumière **Directional Light**.

2.1.1 Directional Light

Supprimer cette **Directional Light**. Ce sera le soleil qui fera office de source de lumière.

2.1.2 Main Camera

Modifier ses propriétés dans l'onglet **Inspector** pour correspondre à l'image de la fig. 3a.



(a) Propriétés de la Main Camera

Solution

Cliquer sur le bouton Add Component en bas de l'Inspector. Dans la barre de recherche, entrer **Skybox**.

On crée ensuite un **Material** personnalisé :

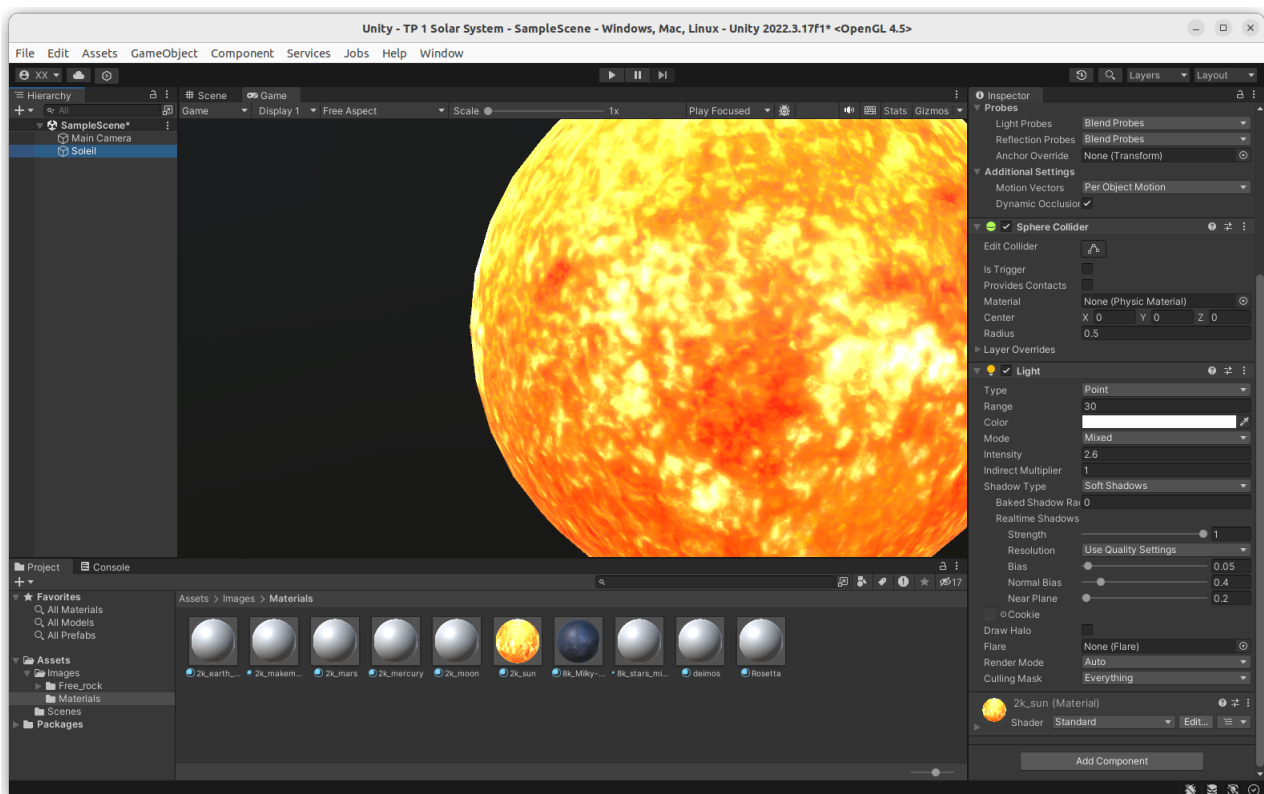
- Dans l'onglet **Project**, bouton droit de la souris sur **Assets > Images > Materials** et sélectionner **Create > Material**.
- Donner un nom (par exemple "*8k_milky-way-2*" au **Material** qui apparaît (l'extension ".mat" est automatiquement ajoutée).
- Dans l'Inspector, les propriétés par défaut du nouveau **Material** sont affichées.
- Cliquer sur le petit cercle à côté de **Albedo** pour ouvrir le choix de textures. Dans la liste qui s'affiche, sélectionner "*8k_milky-way-2.jpg*" (ce fichier doit exister au préalable).
- L'affichage de la sphère de test en bas de l'Inspector passe du blanc par défaut à la texture sélectionnée.
- Sélectionner de nouveau **Main Camera** dans l'onglet **Hierarchy**.
- Dans le composant **Skybox** : cliquer sur le cercle à droite du champ **Custom Skybox** et sélectionner "*8k_milky-way-2*".

Solution

2.2 Soleil

- Dans l'onglet **Hierarchy** : bouton droit > "**3D Object > Sphere**"
- Renommer en "*Soleil*"
- Dans **Inspector** :
 - **Add Component > Light**
 - Modifier les propriétés de **Light**
 - **Range** : 30
 - **Mode** : **Mixed** (voir la doc.)
 - **Intensity** : 2.6

- Shadow type : Soft shadows
 - Dans Project, ouvrir le dossier Images > Materials
 - Vérifier que l'asset "2k_sun" est présent. Si son aspect est blanc (défaut), lui associer dans l'Inspector :
 - Albedo : la texture 2k_sun
 - Emission > Color : la même texture
 - Cliquer sur cet asset et glisser-déposer sur le *Soleil* dans la scène ou l'Inspector : le composant Material se met à jour. Laisser les propriétés par défaut.
- Le résultat devrait ressembler à celui affiché en fig. 4a (en sélectionnant l'onglet Game).



(a) Soleil éclairant la scène

Solution

2.2.1 Terre

Ajouter un objet 3D Sphere dans la Hierarchy comme enfant de *Soleil* et le nommer *Terre*.

- Inspector -> Transform
 - Position : $X = 1.5$; $Y = 0$; $Z = 0$
 - Scale : $X = 0.3$; $Y = 0.3$; $Z = 0.3$ (le redimensionnement est calculé dans le repère local de l'objet parent (ici, *Soleil*))
- Suivre les étapes précédentes utilisées pour affecter une texture au *Soleil* pour affecter une texture à partir de l'image *2k_earth_daymap.jpg*.
- Affecter ce Material directement à *Terre* en le déplaçant depuis Projects sur la vue centrale
- Inspector -> Shader
- Advanced Options : activer Enable GPU instancing et Double Sided Global Illumination
- Au besoin, déplacer *Main Camera* pour englober le *Soleil* et la *Terre* dans la pyramide de vision.

2.2.2 Lune

- Ajouter un objet 3D Sphere dans la hiérarchie comme enfant de *Terre* et le nommer *Lune*.

- **Inspector** -> **Transform**
 - **Position** : X = 0.747 ; Y = 0 ; Z = 0
 - **Scale** : X = .3 ; Y = .3 ; Z = .3
- Reprendre la procédure d'ajout de texture à partir de l'image *2k_moon.jpg*. Si ce **Material** existe déjà, on peut le sélectionner directement en cliquant sur le petit rond dans **Inspector** > **Mesh Renderer** > **Materials**.
- Ajouter une **Camera** comme enfant de *Lune* et laisser les propriétés par défaut, puis ajouter une **Skybox** en reprenant la procédure utilisée pour la *Main Camera*.

ATTENTION : cette seconde caméra est activée par défaut et prend la place de la première dans le **Game View**. Désactiver cette caméra en désactivant la propriété **Inspector** -> **Camera**.

2.2.3 Mars

- Ajouter un objet 3D **Sphere** dans **Hierarchy** comme enfant de *Soleil* et le nommer *Mars*.
- **Inspector** -> **Transform**
 - **Position** : X = 2.27 ; Y = 0 ; Z = 0
 - **Scale** : X = 0.25 ; Y = 0.25 ; Z = 0.25
- Reprendre la procédure d'ajout de texture à partir de l'image *2k_mars.jpg*.

2.2.4 Phobos

On a décrit précédemment comment faire interpréter par **Unity** le fichier relatif à *Phobos*, au format *glTF*. D'autres fichiers sont disponibles sur le site de la NASA par exemple.

- Vérifier que l'objet importé (en fait, une archive) est bien placé dans **Assets** > **Images** et l'intégrer dans **Hierarchy** comme enfant de *Mars*.
- **Inspector** -> **Transform**
 - **Position** : X = -.7 ; Y = 0 ; Z = 0
 - **Scale** : X = 0.01 ; Y = 0.01 ; Z = 0.01

2.2.5 Deimos

Reprendre la procédure précédente pour intégrer *Deimos* comme lune de *Mars*. On peut utiliser le fichier sur ce site.

On a décrit précédemment comment faire interpréter par **Unity** le fichier relatif à *Phobos*, au format *glTF*. D'autres fichiers sont disponibles sur le site de la NASA par exemple.

- Vérifier que l'objet importé (en fait, une archive) est bien placé dans **Assets** > **Images** et l'intégrer dans **Hierarchy** comme enfant de *Mars*.
- **Inspector** -> **Transform**
 - **Position** : X = 0 ; Y = 1 ; Z = 0
 - **Scale** : X = 0.01 ; Y = 0.01 ; Z = 0.01

3 Premier script pour animer tout cela

Je vous propose de réaliser le fameux script **JeTourne.cs** du cours, qui se contente d'appeler la rotation autour du soleil. Chercher dans la documentation **Unity**, le rôle de la fonction **FixedUpdate()**. Modifiez votre programme pour l'utiliser en conséquence comme dans le cours. Nous ferons en particulier les variables suivantes sans les lier à des boutons ou autre widget :

- Ajoutez dans votre script une variable booléenne qui détermine si l'objet en question tourne ou non.
- Placez la vitesse de rotation de votre objet paramètre.
- Surchargez les fonctions intéressantes du cycle de vie de votre objet pour en faire un affichage dans la console.

Solution

On place d'abord le script dans le dossier **Assets**^a. Puis on ajoute ce script à *Soleil* avec un glisser-déposer ou l'Inspector : Add Components > Scripts > Je Tourne.

Contenu de JeTourne.cs :

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class JeTourne : MonoBehaviour
{
    // Par introspection, l'attribut public [mustTurn] est ajoute
    // sous forme de [Toggle] dans les proprietes du script de l'Inspector,
    // sous le nom [Must Turn]
    // Modifier cette valeur dans l'Inspector modifie cet attribut dans ce code
    public bool mustTurn = true;

    // L'attribut [rotationSpeed] est egalement ajoute dans les proprietes du script,
    // sous le nom [Rotation Speed]
    // Modifier cette valeur dans l'Inspector modifie cet attribut dans ce code
    public double rotationSpeed = 5.0;

    // Start is called before the first frame update
    void Start()
    {
        Debug.Log("Start()_de_" + gameObject.name);
        mustTurn = true;
    }

    // Update is called once per frame
    void Update()
    {
        Debug.Log("Update()_de_" + gameObject.name);
    }

    // FixedUpdate is called at fixed frames. To use with rigid Bodies
    // https://docs.unity3d.com/ScriptReference/MonoBehaviour.FixedUpdate.html
    void FixedUpdate()
    {
        Debug.Log("FixedUpdate()_de_" + gameObject.name);
        if (mustTurn)
        {
            Debug.Log("Rotation_de_" + gameObject.name);
            // Rotation autour de l'axe Y
            // transform.Rotate(0, 5 * Time.fixedDeltaTime, 0);
            transform.Rotate(0, ((float)rotationSpeed) * Time.fixedDeltaTime, 0);
        }
    }

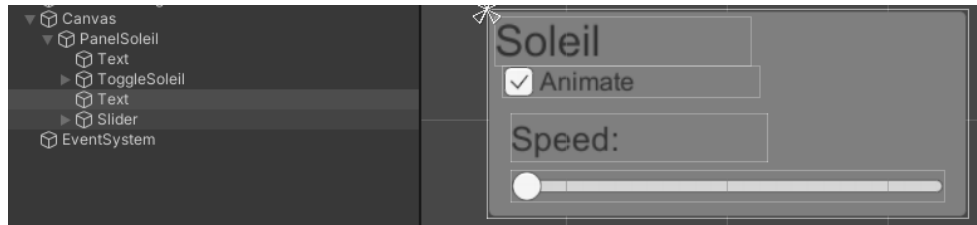
    public void toggle()
    {
        Debug.Log("toggle()_de_" + gameObject.name);
        mustTurn = !mustTurn;
    }
}
```

Les logs sont affichés dans l'onglet **Console**, à côté de **Project**. On constate que tous les éléments enfants de *Soleil* subissent le même mouvement, avec le *Soleil* comme centre de rotation.

a. On peut créer un dossier **Scripts** spécifique, au besoin.

4 Interaction pour le contrôle du système solaire

Réalisez l'interface proposée ci-dessous pour le soleil :



Solution

4.1 Création du canvas

- Dans **Hierarchy** : bouton droit : **UI** -> **Canvas**. Un objet **EventSystem** est également automatiquement ajouté.
- Parmi les propriétés de ce *Canvas*, les valeurs de position et de dimensions sont automatiquement définies et non modifiables. Ces valeurs seront adaptées au contenu de cet objet.
- **Inspector** -> **Rect Transform** (ces valeurs peuvent être différentes selon la configuration de la scène)
 - Pos X = 459 ; Pos Y = 257 ; Pos Z = 0
 - Width = 918 ; Height = 514

4.2 PanelSoleil

- Ajouter un **UI** -> **Panel** nommé *PanelSoleil* comme enfant de *Canvas*.
- **Inspector** -> **Rect Transform**
 - cliquer sur l'icône de positionnement en haut à gauche, **Stretch** / **Stretch** et sélectionner **top** / **left**
 - **Pivot X** = 0 ; **Y** = 1. Ces valeurs représentent le point à partir duquel les coordonnées suivantes sont calculées
 - Pos X = 10 ; Pos Y = -40 ; Pos Z = 0
 - Width = 300 ; Height = 130

On peut modifier ces valeurs, du moment que ce **Panel** se retrouve bien à l'intérieur de *Canvas*. Le panel apparaît sous la forme d'un rectangle gris dans la fig. 5 en vue **Game**.

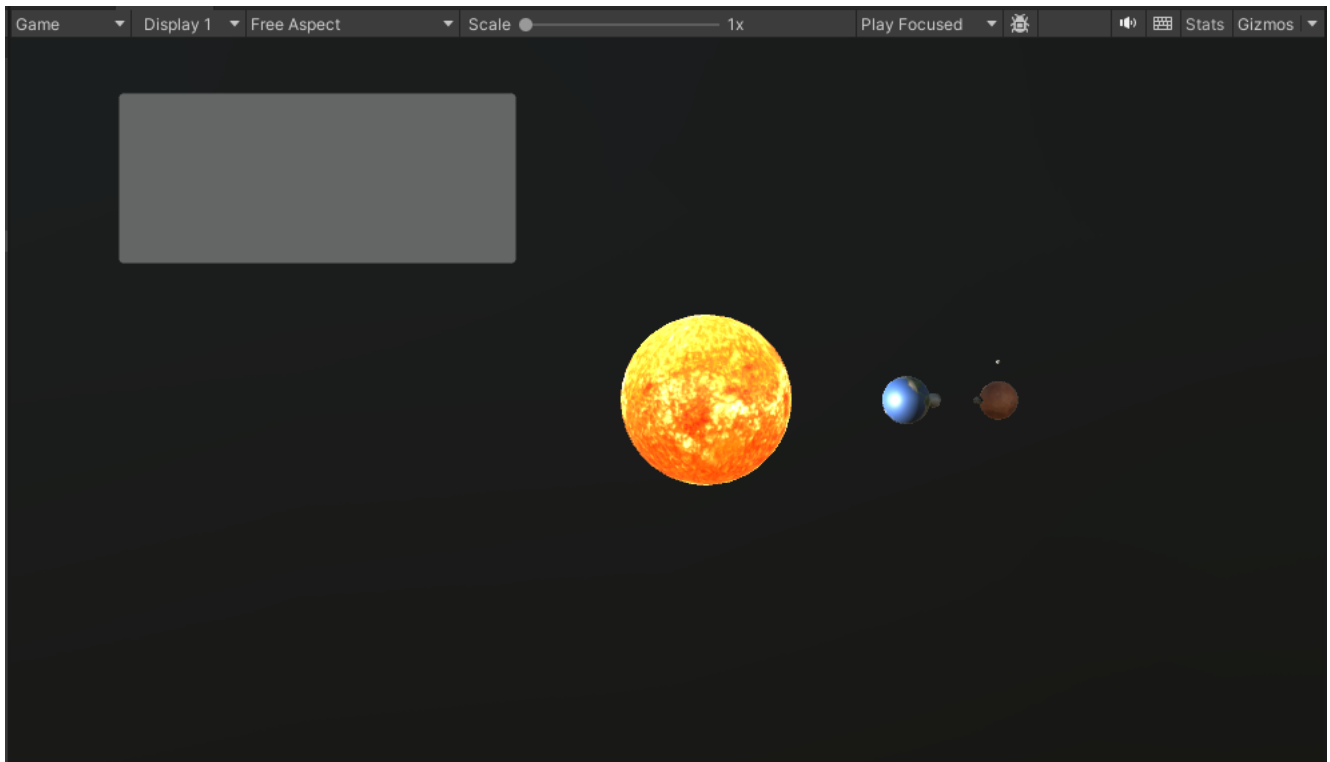


FIGURE 5 – Un panel vide actuellement

Solution

4.2.1 Label du Panel

- Ajouter un UI > Legacy > Text nommé "*NomPanelSoleil*" comme enfant de *PanelSoleil*.
- Inspector > Rect Transform
 - Stretch / Stretch > top / left
 - Pivot X = 0 ; Y = 1
 - Pos X = 10 ; Pos Y = -10 ; Pos Z = 0
 - Width = 160 ; Height = 30
- Inspector > Text
 - Text > *Soleil*
 - Character > Font Size > 24

4.2.2 Case à cocher du Panel

- Ajouter un UI > Toggle nommé "*TogglePanelSoleil*" comme enfant de *PanelSoleil*.
- Inspector > Rect Transform
 - Stretch / Stretch > top / left
 - Pivot X = 0 ; Y = 1
 - Pos X = 10 ; Pos Y = -50 ; Pos Z = 0
 - Width = 160 ; Height = 20

TogglePanelSoleil contient lui-même une hiérarchie :

- Background > Checkmark
- Label : à renommer en "*LabelTogglePanelSoleil*" et modifier sa propriété Text pour remplacer "*Toggle*" par "*Animer*".

4.2.3 Label Vitesse

- Ajouter un UI > Legacy > Text nommé "*VitesseAnimSoleil*" comme enfant de *PanelSoleil*.
- Inspector > Rect Transform
 - Stretch / Stretch > top / left
 - Pivot X = 0 ; Y = 1
 - Pos X = 10 ; Pos Y = -80 ; Pos Z = 0
 - Width = 160 ; Height = 20

4.2.4 Slider Vitesse

- Ajouter un UI > Slider nommé "*SliderVitesseAnimSoleil*" comme enfant de *PanelSoleil*.
- Inspector > Rect Transform
 - Stretch / Stretch > top / left
 - Pivot X = 0 ; Y = 1
 - Pos X = 10 ; Pos Y = -100 ; Pos Z = 0
 - Width = 250 ; Height = 20
- Inspector > Slider
 - Min Value = 0
 - Max Value = 100

4.3 Panel Terre et Panel Mars

- Dupliquer *PanelSoleil* avec UI -> Duplicate dans la Hierarchy en deux exemplaires : *PanelTerre* et *PanelMars*
- Vérifier que les propriétés des objets dupliqués sont bien mises à jour (par exemple pour les valeurs min et max des Sliders)
- Déplacer ces Panels les uns en-dessous (ou à côté) des autres dans le Canvas principal
- Renommer les objets en conséquence

À ce stade, la vue en mode Game devrait ressembler à la fig. 6.

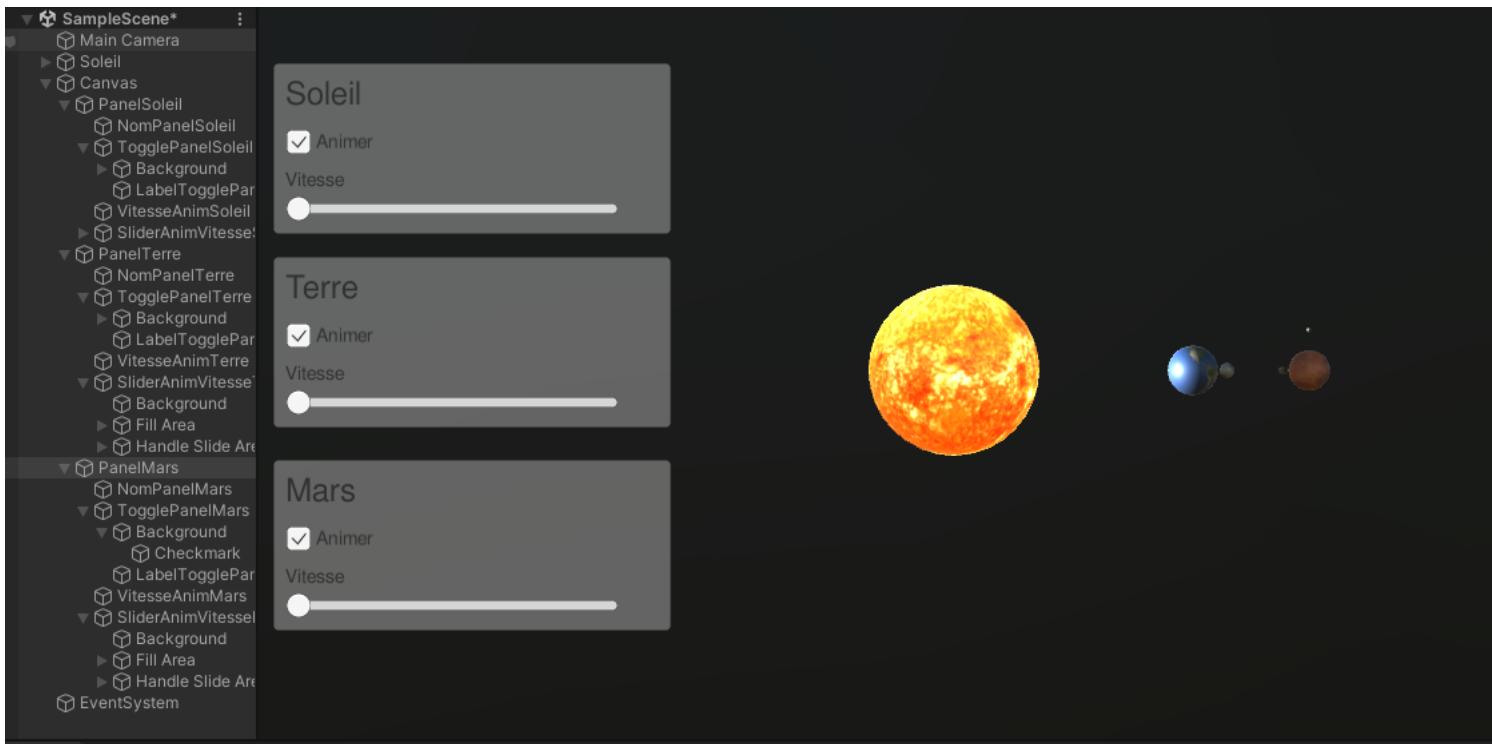


FIGURE 6 – Les trois panels

Ensuite sur ce même modèle, réalisez l'interface pour la *Terre* et *Mars* uniquement. Maintenant que vous avez une jolie interface, vous allez réaliser la connexion au écouteur d'événement des boutons d'animation pour contrôler si la rotation de l'astre associé est active ou non. En bref, il vous est demandé de réaliser une fonction, qui détecte les changements de valeur de la **Checkbox** pour en activer/désactiver l'animation selon la valeur du booléen. Pour Le **Slider**, nous considérons qu'il pourra prendre des valeurs de 0 à 100 et qui contrôlera la vitesse de rotation de l'objet associé.

Solution

Dans un premier temps :

1. Affecter le même script `JeTourne.cs` à la *Terre* et *Mars* avec **Inspector** > **Add Component** et la barre de recherche.
2. Dans le mode **Game**, désactiver à tour de rôle le **Toggle Must Turn** de chaque **GameObject** pour mieux observer ses effets.

Ensuite, associer le script suivant nommé `PanelsoleilScript.cs` au *PanelSoleil* avec **Inspector** > **Script** > **New Script**^a. Il permet de modifier *via* ce **Canvas** l'activation/désactivation de la rotation du *Soleil* et sa vitesse. Pour éditer ce script depuis l'**Inspector**, cliquer sur les 3 points verticaux à droite de ce composant pour faire apparaître un menu contextuel, puis cliquer sur **Edit**.

```
// Inspire de
// https://docs.unity3d.com/2019.1/Documentation/ScriptReference/UI.Toggle-onValueChanged.html
// Attach this script to a Toggle GameObject. To do this, go to Create>UI>Toggle.
// Set your own Text in the Inspector window
using UnityEngine;
using UnityEngine.UI;
public class PanelSoleilScript : MonoBehaviour
{
    Toggle m_Toggle;
    Slider m_Slider;
    GameObject soleil;
    void Start()
    {
        // m_Toggle est le Toggle enfant de PanelSoleil
        m_Toggle = GetComponentInChildren<Toggle>();
        if (m_Toggle == null)
            Debug.Log("m_Toggle_LeSoleil_=_nul");
        // m_Slider est le Slider enfant de PanelSoleil
        m_Slider = GetComponentInChildren<Slider>();
        if (m_Slider == null)
            Debug.Log("m_Slider_LeSoleil_=_nul");
        // Add listener for when the state of the Toggle changes, to take action
        m_Toggle.onValueChanged.AddListener(delegate {
            ToggleValueChanged(m_Toggle);
        });
        m_Slider.onValueChanged.AddListener(delegate {
            SliderValueChanged(m_Slider);
        });
        soleil = GameObject.Find("Soleil");
    }
    void ToggleValueChanged(Toggle change)
    {
        soleil.GetComponent<JeTourne>().mustTurn =
            !soleil.GetComponent<JeTourne>().mustTurn;
    }
    void SliderValueChanged(Slider change)
    {
        soleil.GetComponent<JeTourne>().rotationSpeed = change.value;
    }
}
```

- Reprendre le script ci-dessus et modifier les occurrences de *Soleil* par *Terre* ou *Mars*.
- Une fois ces scripts compilés, les ajouter respectivement à *PanelTerre* et *PanelMars*.
- Ne pas oublier de passer la Max Value de *SliderVitesseAnimTerre* et *SliderVitesseAnimMars* à 100 si nécessaire.

a. Il suffit d'indiquer le nom du script, sans son extension

5 Placement de caméra

Lisez la page du guide suivant : <https://docs.unity3d.com/Manual/MultipleCameras.html>. Elle explique les deux façons de contrôler l'affichage d'une caméra simplement.

Pour mettre en pratique les explications du lien, nous vous proposons d'ajouter une caméra locale à la *Lune* (de la *Terre*). Ajoutez des widgets à l'interface pour changer de caméra à la volée comme indiquez dans la documentation.

Pour les explications de la seconde partie, on se propose d'ajouter une 3ème caméra en vue de haut de notre système

solaire. Cette vue devra apparaître en miniature dans le bord bas gauche de la fenêtre (ou un autre bord si vous préférez).

Solution

5.1 Ajout d'une Caméra pour la Lune

- Lorsque plusieurs caméras sont utilisées, la caméra visible est celle dont la valeur de la propriété `depth` est la plus élevée.
- Si ce n'est déjà fait, ajouter un `GameObject Camera Camera Lune` à la *Lune*.
- Ajouter un `Component SkyBox` utilisant "*8k_stars_milky_way*"
- Décocher l'*Audio Listener* de cette caméra (il est automatiquement ajouté à chaque nouvelle caméra, mais il ne devrait y en avoir qu'un seul activé à la fois dans la scène).

5.2 Ajout d'un Panel et du script

- Dans le `Canvas` principal, ajouter un `Panel` nommé `PanelCameras` qui comprendra un `Toggle` nommé *ToggleCameras*, dont le `Label` contiendra le texte "*Main Camera / Camera Lune*".
- Aligner *PanelCameras* avec les autres `Panels` de *Canvas*.
- Associer le script suivant *PermutationCameraScript.cs* à *PanelCameras*.

```

using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;
// Inspire de https://docs.unity3d.com/Manual/MultipleCameras.html
public class PermutationCamerasScript : MonoBehaviour
{
    // Toggle du Panel associe a ce script
    Toggle m_Toggle;
    // Cameras manipulees
    Camera mainCamera;
    Camera moonCamera;
    // Start is called before the first frame update
    void Start()
    {
        // m_Toggle est le Toggle enfant de PabelCameras
        m_Toggle = GetComponentInChildren<Toggle>();
        if (m_Toggle == null)
            Debug.Log("m_Toggle_PanelCameras=_nul");
        //Add listener for when the state of the Toggle changes, to take action
        m_Toggle.onValueChanged.AddListener(delegate {
            ToggleValueChanged(m_Toggle);
        });
        mainCamera = getCameraFromParentName("Main_Camera");
        mainCamera.enabled = true;
        moonCamera = getCameraFromParentName("Camera_Lune");
        moonCamera.enabled = false;
    }
    private Camera getCameraFromParentName(System.String parentName) {
        GameObject cameraObject = GameObject.Find(parentName);
        if (cameraObject == null)
            Debug.Log("CameraObject_of_ " + parentName + "_nul");
        Camera camera = cameraObject.GetComponent<Camera>();
        if (camera == null)
            Debug.Log("Camera_of_ " + parentName + "_nul");
        return camera;
    }
    void ToggleValueChanged(Toggle change)
    {
        mainCamera.enabled = !mainCamera.enabled;
        moonCamera.enabled = !moonCamera.enabled;
    }
}

```

5.3 3ème caméra

- Dans la Hierarchy, ajouter une Camera et la nommer "*Camera Up View*".
- La placer et l'orienter de manière à offrir une "vue de dessus" du système solaire, par exemple, dans l'Inspector :
 - Transform > Position -> X = 0 ; Y = 15 ; Z = -0.8
 - Transform > Rotation -> X = 84.43 ; Y = -20.639 ; Z = -124.044
 - Camera > Field of View = 28
 - Depth = 2 : valeur supérieure à la propriété de même nom des autres caméras
 - ViewPort Rect > X = 0.8 ; Y = 0 : place le rectangle de cette vue en bas à droite de la scène finale
 - ViewPort Rect > W = 0.2 ; H = 0.2 : facteurs d'échelle des dimensions du rectangle de vue

- Ajouter un **Component SkyBox** utilisant "*8k_stars_milky_way*".
- Décocher l'*Audio Listener* de cette caméra.

À ce stade, l'interface ressemble à celle de la fig. 7 (la vue de **Camera Up View** est affichée en bas à droite de l'écran)

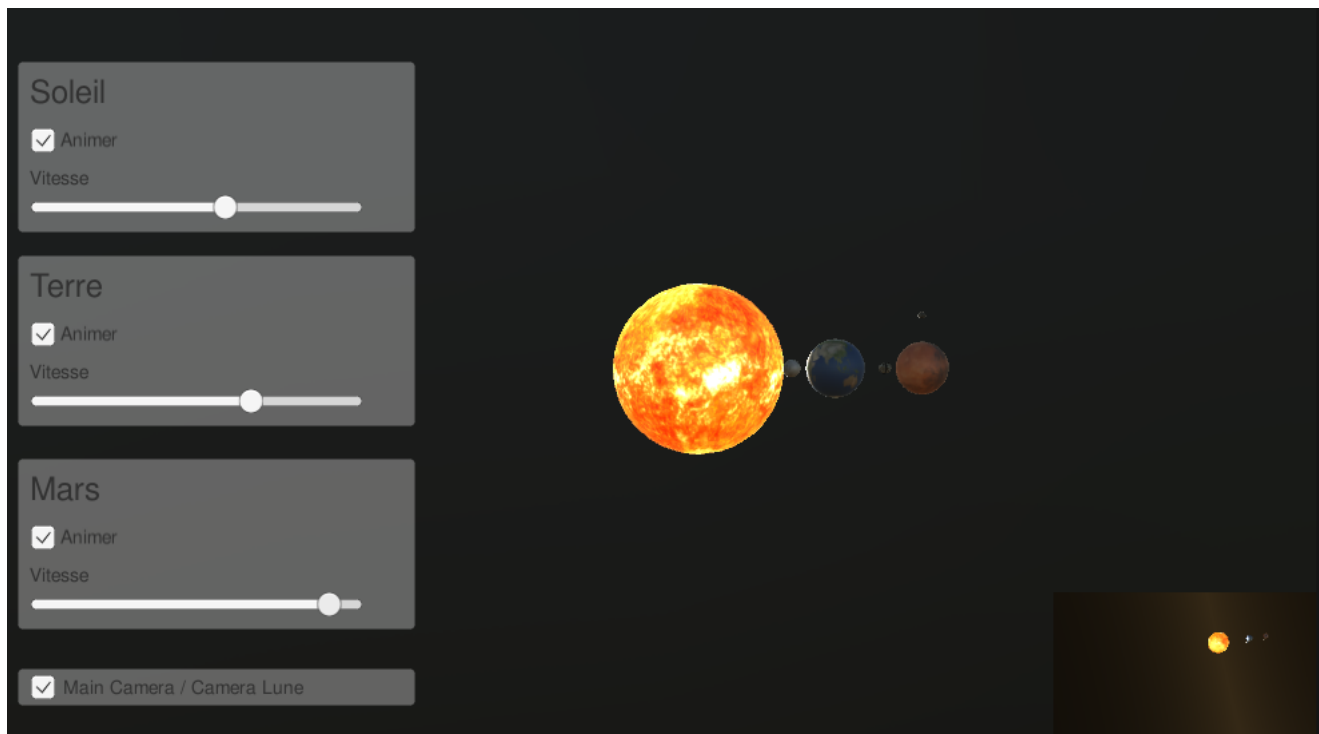


FIGURE 7 – Interface avec **Camera Up View**

6 Pour aller plus loin

La mécanique céleste mise en place jusqu'à présent est vraiment catastrophique, mais offre un cadre suffisant pour l'apprentissage des mécaniques. Nous vous proposons d'ajouter une comète cyclique (ou autre astre) qui n'est plus sous-fils du Soleil, mais bien un objet quelconques.

Pour pouvoir animer votre comète vous allez devoir lui fournir une équation qui va traduire sa trajectoire pour cela je vous propose de vous inspirer du lien suivant https://fr.wikipedia.org/wiki/Trajectoire_d%27une_com%C3%A8te ou d'utiliser l'équation d'un cercle autour du soleil par exemple.

Solution

L'objectif est de créer un objet qui tourne autour du *Soleil* mais n'appartient pas à sa hiérarchie.

- Importer un objet 3D au format **.obj** pour représenter un nouvel astéroïde, par exemple Benuu ^a.
- Placer cet objet dans le **Project > Assets > Images**.
- Déplacer cet objet depuis **Project** vers **Hierarchy**, ce qui l'ajoute à la scène. Sa position initiale est en $(X=0, Y=0, Z=0)$: le déplacer sur un autre axe, par exemple en $(X=2.2, Y=1.5, Z=0)$.
- Cet objet contient un enfant nommé "*default*" : associer à ce dernier un enfant de type **Trail** : bouton **droit > Effects > Trail**.
- Modifier la Position de ce **Trail** en $(X=0; Y=0; Z=0)$ pour le "coller" sur *default*.
- Modifier le **Trail Renderer** de ce **Trail** :
 - en diminuant sa largeur **Width** (valeur *0.2* par exemple);
 - en modifiant éventuellement sa propriété **Color**;
 - en modifiant la propriété **Time** correspondant à la durée de vie de la trace laissée par l'objet.

Le script suivant `TrajectoiresScript.cs` associé à *Bennu* permet de créer une trajectoire elliptique autour du centre du repère global, avec le soleil constituant l'un des foyers de l'ellipse. Les attributs `Semi Major` et `Ecliptic Angle` de *Bennu* accessibles depuis l'Inspector permettent de changer la forme de l'ellipse (un cercle pour `Ecliptic Angle = 0`, une ellipse de plus en plus allongée lorsque `Ecliptic Angle` tend vers `Semi Major`).

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UIElements;

// Trajectoire circulaire ou elliptique en utilisant le soleil comme foyer d'une ellipse

public class TrajectoiresScript : MonoBehaviour
{
    private GameObject soleil;

    public float semiMajor = 1f ; // demi-grand axe de l'ellipse > 0

    // angle de rotation entre ce GameObject et le soleil ;
    // a calculer a chaque pas de temps
    public float angle = 0f; // a convertir en radians

    // inclinaison de l'ellipse (en degres) en restant dans le meme plan de
    // l'ecliptique
    public float eclipticAngle = 0f; // a convertir en radians

    // L'excentricite est definie selon la distance entre le centre de l'ellipse,
    // l'un de ses foyers et le demi-grand axe
    private float eccentricity;

    // Start is called before the first frame update
    void Start()
    {
        soleil = GameObject.Find("Soleil");
        if (soleil == null) {
            Debug.Log("TrajectoiresScript: _soleil = _null");
        }
    }
    // Update is called once per frame
    void Update() { }
```



```

// FixedUpdate is called at fixed frames. To use with rigid Bodies
// https://docs.unity3d.com/ScriptReference/MonoBehaviour.FixedUpdate.html
void FixedUpdate() {
    // En toute generalite, l'excentricite est definie sur un axe,
    // X par convention.
    // Formule generale :
    // excentricite = Abs(centre de l'ellipse.X - foyer de l'ellipse.X) /
    //   demi-grand axe
    // => et cette valeur doit etre comprise dans [0 ; 1[

    // On suppose ici que le centre de l'ellipse est le centre du
    // repere global (0,0,0)
    eccentricity = Mathf.Abs(soleil.transform.position.x) / semiMajor;
    Debug.Log("eccentricity=" + eccentricity);

    // semiMinor = demi-petit axe de l'ellipse
    //=> semiMajor et semiMinor sont lies a l'excentricite de l'ellipse E
    // par semiMinor = semiMajor * sqrt(1 - E*E)

    // => si semiMajor et semiMinor sont egales, l'orbite est un cercle
    // (facile a voir avec tiltEclipticAngle = 0)
    float semiMinor = semiMajor * Mathf.Sqrt(1 - eccentricity * eccentricity);
    float angleRad = Mathf.Deg2Rad * angle;
    float angleRadCos = Mathf.Cos(angleRad);
    float angleRadSin = Mathf.Sin(angleRad);
    float eclipticAngleRad = Mathf.Deg2Rad * eclipticAngle;
    float eclipticAngleRadCos = Mathf.Cos(eclipticAngleRad);
    float eclipticAngleRadSin = Mathf.Sin(eclipticAngleRad);

    // Remarque: Multiplier par Time.fixedDeltaTime (=0.02) donne des
    // resultats proches de 0
    // les coordonnees X et Z resultantes sont donc proches de 0 =>
    // ce GameObject ne se deplace donc quasiment pas
    // => on "equilibre" les coordonnees en multipliant par 10
    this.gameObject.transform.position =
    new Vector3(((semiMajor * angleRadCos * eclipticAngleRadCos) -
    (semiMajor * angleRadSin * eclipticAngleRadSin)) *
    Time.fixedDeltaTime * 10f,
    0,
    ((semiMinor * angleRadCos * eclipticAngleRadSin) -
    (semiMinor * angleRadSin * eclipticAngleRadCos)) *
    Time.fixedDeltaTime * 10f);

    angle += 1f; //can be used as speed

    if (angle > 360f) { angle = 0f; }

    Debug.Log("new X=" + this.gameObject.transform.position.x +
    ", new Z=" + this.gameObject.transform.position.z);

    Debug.Log("soleil.X=" + soleil.transform.position.x);
    // On peut faire varier la position du Soleil en X,
    // sans depasser la valeur de semiMajor

    Debug.Log("Time.fixedDeltaTime=" + Time.fixedDeltaTime);
}
}

```

La fig. 8 affiche la trace laissée par **Bennu** en cours d'animation.

a. Ce maillage n'a pas de normale incluses, mais **Unity** les recalcule automatiquement.

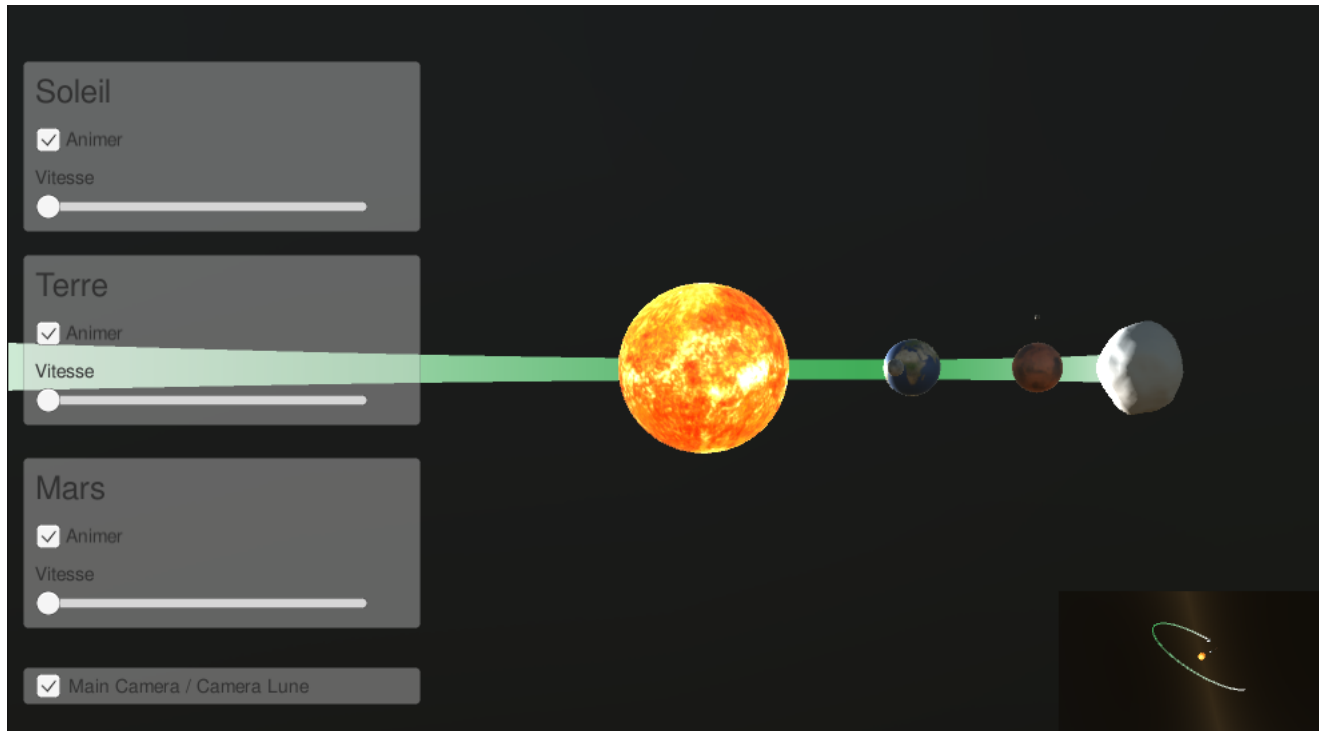


FIGURE 8 – Interface avec **Bennu** en mouvement

7 Toujours plus loin

Si vous avez tout fini, nous pouvons complexifier les traitements en regardant la documentation :

- En utilisant la fonction `LookAt` dans la classe `Transform`, ajoutez pour chaque interface un bouton 'Center View' qui centre la vue de la caméra principale vers l'astre associé.
- Modifier votre traitement précédent pour que la caméra tourne doucement vers sa position finale pour éviter de perdre l'utilisateur ou lui donner des nausées. Pour cela, nous utiliserons une interpolation linéaire ou sphérique entre le point de vue de départ et le point de vue d'arrivée avec une vitesse constante.

Solution

7.1 Création des Toggles

L'objectif est de pouvoir sélectionner la planète sur laquelle la vue de la caméra principale sera centrée, ou bien de revenir à l'orientation originale de la caméra.

- Repositionner les `Panel`s du `Canvas` principal pour ajouter un nouveau `Panel` nommé "*PanelLookAt*".
- Ajouter dans *PanelLookAt* un `Label` nommé *NomPanelLookAt* et contenant le `Text` "*Cameras LookAt*".
- Ajouter une succession de `Toggles` nommés respectivement *ToggleSoleil*, *ToggleTerre*, *ToggleMars* et *ToggleOriginal* avec les `Text` de leurs `Labels` respectifs : *Soleil*, *Terre*, *Mars* et *Original*.
- Pour chaque `Toggle` :
 - Désactiver la propriété `IsOn`, sauf pour *Original*
 - Ajouter le `Component Layout Element` et spécifier la propriété `LayoutElement -> Preferred Height = 20` (coche activée).

7.2 ToggleGroup

On va maintenant regrouper tous ces `Toggle` dans un `ToggleGroup`.

- Dans *PanelLookAt*, créer un `Create Empty` nommé *ToggleGroupLookAt*. Il s'agit d'un `GameObject` quasiment vide avec des propriétés minimales, que l'on va utiliser comme un conteneur. Le redimensionner au besoin pour qu'il tienne dans *PanelLookAt*.
- Modifier la hiérarchie pour que *ToggleGroupLookAt* devienne le parent des `Toggles`.
- Ajouter à *ToggleGroupLookAt* le `Component Toggle Group` et désactiver le booléen `Allow Switch Off` de ce dernier pour faire en sorte qu'un seul `Toggle` enfant soit sélectionné à la fois.
- Ajouter le `Component Vertical Layout Group` à *ToggleGroupLookAt* si les `Toggles` sont disposés verticalement ; le `Component Horizontal Layout Group` sinon.
- Dans ce `Component`, désactiver la propriété `Child Force Expand > Height` pour `Vertical Layout Group` ou `Child Force Expand > Width` pour `Horizontal Layout Group`.
- Ajouter le script suivant *ToggleGroupLookAtScript.cs* à *ToggleGroupLookAt*. Il permet de récupérer la planète associée au `Toggle` sélectionné.

```

using UnityEngine;
using System.Collections;
using UnityEngine.UI;
using System.Linq;
public class ToggleGroupLookAtScript : MonoBehaviour {
    ToggleGroup m_ToggleGroup;
    Toggle m_PreviousToggle;
    Toggle m_SelectedToggle;

    public Toggle GetCurrentSelection() {
        return m_ToggleGroup.ActiveToggles().FirstOrDefault();
    }

    public Toggle GetPreviousSelection() {
        return m_PreviousToggle;
    }

    void Start() {
        m_ToggleGroup = GetComponent<ToggleGroup>();
        m_PreviousToggle = GetCurrentSelection();
        m_SelectedToggle = GetCurrentSelection();
        Debug.Log("[ToggleGroupLookAtScript] [Start]_m_PreviousToggle=_ " +
            m_PreviousToggle.name + ",_m_SelectedToggle=_ " +
            m_SelectedToggle.name);
    }

    public GameObject GetGameObjectFromToggle(Toggle toggle) {
        switch(toggle.name) {
            case "ToggleSoleil":
                return GameObject.Find("Soleil");
            case "ToggleTerre":
                return GameObject.Find("Terre");
            case "ToggleMars":
                return GameObject.Find("Mars");
            case "ToggleOriginal":
                return GameObject.Find("Main_Camera");
            default:
                return null;
        }
    }

    void Update() {
        if (GetCurrentSelection() != m_SelectedToggle) {
            m_PreviousToggle = m_SelectedToggle;
            m_SelectedToggle = GetCurrentSelection();

            Debug.Log("[ToggleGroupLookAtScript] [Update]_m_PreviousToggle=_ " +
                m_PreviousToggle.name + ",_m_SelectedToggle=_ " +
                m_SelectedToggle.name);
        }
    }
}

```

Sélectionner l'ensemble des **Toggle** enfants, puis cliquer sur **ToggleGroupLookAt** et le déplacer à la souris dans la propriété **Toggle > Group** des enfants sélectionnés. La valeur de cette propriété passe de **None (Toggle Group)** à **ToggleGroupLookAt (Toggle Group)**.

On peut tester l'activation des boutons en démarrant la scène et en cliquant sur chaque bouton du groupe, ce qui désactive le bouton précédent.

7.3 Changement de caméra

Le script *LookAtPlanetSlerpScript.cs* ci-dessous est attaché à chaque **Toggle** enfant de **ToggleGroupLookAt**. Pour chacun d'eux, il vérifie s'il est sélectionné et dans ce cas, si le **Toggle** précédemment sélectionné est différent, change la position de la caméra d'une planète à l'autre, selon une interpolation sphérique basée sur les quaternions.

```
// Script attache aux GameObject "Toggle" enfants de ToggleGroupLookAt pour
// centrer la vue de la camera principale sur une planete donnee,
// ou revenir a l'orientation originale.

// Attention : chaque Toggle possede sa propre instance de ce script.
// Donc toutes les variables du script sont definies independamment pour chaque Toggle
// (elles ne sont pas partagees)

// Inspire de
// https://docs.unity3d.com/2019.1/Documentation/ScriptReference/UI.Toggle-onValueChanged.html
// Set your own Text in the Inspector window

using UnityEngine;
using UnityEngine.UI;
using System.Collections;
using System.Linq;
public class LookAtPlanetScriptSlerp : MonoBehaviour
{
    Toggle m_Toggle;

    GameObject m_MainCamera;

    float m_TimeCount = 0.0f;

    bool m_EnableInterpolation = false;

    ToggleGroup m_ToggleGroup;

    Quaternion m_MainCameraInitialRotation;

    public ToggleGroupLookAtScript m_ToggleGroupLookAtScript;

    void Start() {
        Debug.Log("[LookAtPlanetSlerpScript]_1");

        // On recupere le script identifiant la planete vers laquelle deplacer la camera
        m_ToggleGroupLookAtScript =
            GameObject.FindObjectOfType(typeof(ToggleGroupLookAtScript))
                as ToggleGroupLookAtScript;

        Debug.Log("[LookAtPlanetSlerpScript]_2");

        // ToggleGroup parent
        m_ToggleGroup = GetComponentInParent<ToggleGroup>();
        if (m_ToggleGroup == null) {
            Debug.Log("[LookAtPlanetSlerpScript]_m_ToggleGroup=null");
        }
        else {
            Debug.Log("[LookAtPlanetSlerpScript]_m_ToggleGroup=" + m_ToggleGroup.name);
        }

        // m_Toggle est le Toggle "enfant" de ce GameObject (en fait le Toggle actuel)
        m_Toggle = GetComponentInChildren<Toggle>();

        if (m_Toggle == null)
            Debug.Log("[LookAtPlanetSlerpScript]_m_Toggle=null");
    }
}
```

```

// Add listener for when the state of the Toggle changes, to take action
m_Toggle.onValueChanged.AddListener(delegate {
    ToggleValueChanged(m_Toggle);
});

// On recherche la [Main Camera] pour la position originale
m_MainCamera = GameObject.Find("Main_Camera");

// Enregistrement de la rotation de la camera originale
m_MainCameraInitialRotation = m_MainCamera.transform.rotation;
if (m_MainCamera == null) {
    Debug.Log("[LookAtPlanetSlerpScript]_m_MainCamera=_nul");
}
}

void ToggleValueChanged(Toggle change) {
    if (m_Toggle.isOn) {
        Debug.Log("[LookAtPlanetSlerpScript]_m_Toggle=_ " + m_Toggle.isOn +
            "_pour_" + name);
        m_EnableInterpolation = true;
    }
}

void FixedUpdate() {
    Toggle selectedToggle = m_ToggleGroupLookAtScript.GetCurrentSelection();
    Toggle previousToggle = m_ToggleGroupLookAtScript.GetPreviousSelection();

    if (m_Toggle.isOn) { // Ce Toggle est active

        // Recuperation de la planete associee
        GameObject selectedObject =
            m_ToggleGroupLookAtScript.GetGameObjectFromToggle(selectedToggle);

        if (m_EnableInterpolation) {
            if (selectedToggle != previousToggle) {
                if (selectedObject != m_MainCamera) {
                    GameObject previousObject =
                        m_ToggleGroupLookAtScript.GetGameObjectFromToggle(previousToggle);

                    if (selectedObject == null || previousObject == null) {
                        Debug.Log("[LookAtPlanetSlerpScript] [Update]_selectedObject=_nul_ou_"
                            + "previousObject=_nul");
                    }
                }
            }
        }

        // https://docs.unity3d.com/2022.2/Documentation/ScriptReference/Quaternion.LookRotation.html
        Vector3 relativePos = selectedObject.transform.position -
            m_MainCamera.transform.position;

        Quaternion rotation = Quaternion.LookRotation(relativePos, Vector3.up);

        // https://docs.unity3d.com/2022.2/Documentation/ScriptReference/Quaternion.Slerp.html
        // [m_timeCount] varie entre 0 (axe de rotation initial) et 1 (axe de rotation final)
        // Sa valeur est mise a jour a chaque frame.
        m_MainCamera.transform.rotation =
            Quaternion.Slerp(m_MainCamera.transform.rotation,
                rotation,
                m_TimeCount);
    }
}

```

```

    else { // selectedObject == m_MainCamera
        GameObject previousObject =
            m_ToggleGroupLookAtScript.GetGameObjectFromToggle(previousToggle);

        if (selectedObject == null || previousObject == null) {
            Debug.Log("[LookAtPlanetSlerpScript] [Update] _selectedObject=_nul_ou_"
                + "previousObject=_nul");
        }

        m_MainCamera.transform.rotation =
            Quaternion.Slerp(m_MainCamera.transform.rotation,
                m_MainCameraInitialRotation,
                m_TimeCount);
    }
    UpdateTimeCount();
}
}
else if (m_MainCamera != selectedObject) {
    m_MainCamera.transform.LookAt(selectedObject.transform);
}
}
}

void UpdateTimeCount() {
    Debug.Log("[LookAtPlanetSlerpScript] [Update] _rotation=_ " +
        m_MainCamera.transform.rotation);
    m_TimeCount += Time.deltaTime;

    // [m_TimeCount] doit rester dans l'intervalle [0; 1].
    if (m_TimeCount > 1.0f) {
        m_TimeCount = 0.0f;

        // On est arrive sur l'axe de rotation final : l'interpolation stoppe
        m_EnableInterpolation = false;
    }
}
}
}

```

La fig. 9 illustre l'animation avec la caméra centrée sur *Mars*.

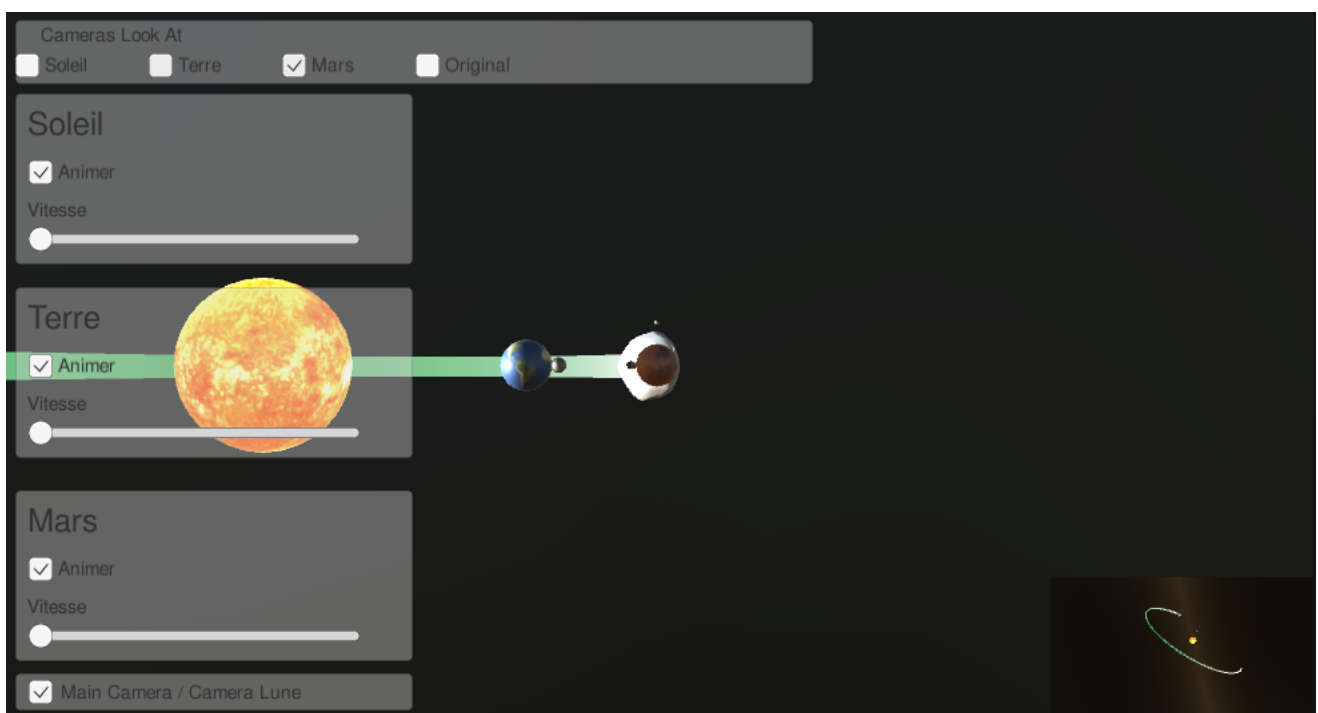


FIGURE 9 – Caméra centrée sur Mars